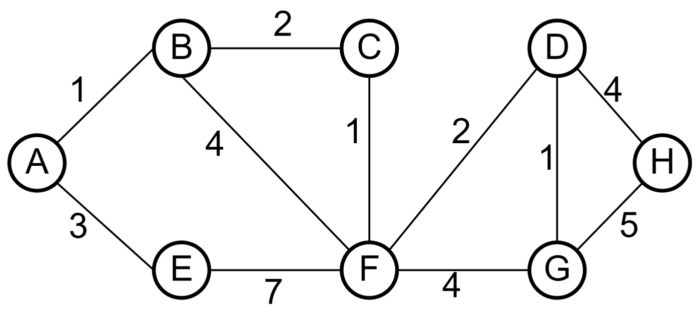
CS 61B — Spring 2019

DFS, BFS, Shortest Paths

Exam Prep 10: April 1, 2019

1 DFS, BFS, Dijkstra's, A*

For the following questions, use the graph below and assume that we break ties by visiting lexicographically earlier nodes first.



(a) Give the depth first search preorder traversal starting from vertex A.

(b) Give the depth first search postorder traversal starting from vertex A.

(c) Give the breadth first search traversal starting from vertex A.

(d) Give the order in which Dijkstra's Algorithm would visit each vertex, starting from vertex A. Sketch the resulting shortest paths tree.

(e) Give the path A* search would return, starting from A and with G as a goal. Let $h(u, v)$ be the valued returned by the heuristic for nodes u and v.

u	v	$h(u, v)$
A	G	9
B	G	7
C	G	4
D	G	1
E	G	10
F	G	3
H	G	5

2 Cycle Detection

Given an undirected graph, provide an algorithm that returns true if a cycle exists in the graph, and false otherwise. Also, provide a Θ bound for the worst case runtime of your algorithm. You may use either an adjacency list or an adjacency matrix to represent your graph.

3 Conceptual Shortest Paths

Answer the following questions regarding shortest path algorithms for a **weighted, undirected graph**. If the question is T/F and the statement is true, provide an explanation. If the statement is false, provide a counterexample.

(a) (T/F) If all edge weights are equal and positive, breadth-first search starting from node A will return the shortest path from a node A to a target node B.

(b) (T/F) If all edges have distinct weights, the shortest path between any two vertices is unique.

(c) (T/F) Adding a constant positive integer k to all edge weights will not affect any shortest path between vertices.

(d) Draw a weighted graph (could be directed or undirected) where Dijkstra's would incorrectly give the shortest paths from some vertex.

4 Shortest Path Algorithm Design *Extra*

Design an efficient algorithm for the following problem: Given a weighted, undirected, and connected graph G, where the weights of every edge in G are all integers between 1 and 10, and a starting vertex s in G, find the distance from s to every other vertex in the graph (where the distance between two vertices is defined as the weight of the shortest path connecting them).

Your algorithm must run asymptotically faster than Dijkstra's. *Hint*: Think about the different graph traversals that you learned in lecture.